Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tadej Razboršek

Vizualizacija aktivnosti v oblaku na platformi OpenStack

diplomsko delo
na univerzitetnem študiju

Mentor: doc. dr. Mojca Ciglarič

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Analizirajte področje računalništva v oblaku, pri tem se osredotočite na storitveni model IaaS - infrastruktura kot storitev in na ogrodje OpenStack. Opišite OpenStackov projekt Ceilometer, ki je namenjen za beleženje podatkov o aktivnosti oblaka. Utemeljite potrebo po podrobnem nadzoru uporabe virov v oblaku in o zaračunavanju porabe z visoko stopnjo drobnozrnatosti. Preučite in primerjajte različna orodja za nadzor in merjenje storitev oblaka, tudi tista, ki so vezana na konkretne ponudnike javnih oblakov. Izpostavite pomanjkljivosti znanih orodij, ki se lahko uporabljajo v oblaku na platformi OpenStack. Nato zasnujte lastno aplikacijo, ki bo omogočala uporabo podatkov, ki jih zbere Ceilometer, in njihovo vizualizacijo po željah uporabnika. Aplikacijo tudi implementirajte, preizkusite in komentirajte njeno delovanje. V zaključku navedite možnosti za nadaljnje delo na tem področju.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Tadej Razboršek, z vpisno številko **63080132**, sem avtor diplomskega dela z naslovom:

Vizualizacija aktivnosti v oblaku na platformi OpenStack

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Mojce Ciglarič,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 7. maja 2014 Podpis avtorja:

Zahvaljujem se mentorici doc. dr. Mojci Ciglarič za vodenje pri izdelavi diplomskega dela in as. Mihi Groharju za svetovanje med izdelavo spletne aplikacije.

Rad bi se zahvalil tudi družini za podporo skozi vsa leta študija.

Kazalo

Povzetek

Računalništvo v oblaku postaja vse bolj priljubljeno in razširjeno. Eden od ključnih elementov vsakega sistema v oblaku je tudi nadzor nad količino porabljenih virov. Ponudnikom omogoča pravično zaračunavanje storitev, uporabnikom pa daje pregled nad aktivnostjo njihovih aplikacij. V okviru diplomskega dela smo implementirali spletno aplikacijo, ki vizualizira podatke o porabi računalniških virov v oblaku, ustvarjenem s pomočjo prosto dostopne programske rešitve OpenStack. V delu najprej definiramo glavne pojme, potrebne za uvrstitev aplikacije na ustrezno mesto v področju računalništva v oblaku. Nato povzamemo njene funkcionalnosti in na koncu opišemo način implementacije. Aplikacija uporablja podatke, ki jih zbere komponenta Ceilometer. Od komponente jih pridobi s komunikacijo preko REST API ter jih hkrati ustrezno filtrira in vizualizira.

Ključne besede: računalništvo v oblaku, nadzorovanje, vizualizacija podatkov, OpenStack, Ceilometer

Abstract

Cloud computing is becoming ever more popular and widely used. A key component of every cloud-computing system is resource monitoring. Suppliers can charge a fair price for the use of their services and the users have better overview of the activity of their applications. In the scope of this diploma thesis, we have implemented a web application that visualizes resource usage in a cloud built using the open-source platform OpenStack. First, we define the main concepts needed to classify the application in the field of cloud computing. We then offer a summary of its functionalities and describe its implementation. The application receives its data from a software component called Ceilometer. The data is retrieved via REST API, and simultaneously filtered and visualized.

Keywords: cloud computing, monitoring, data visualization, OpenStack, Ceilometer

Poglavje 1

Uvod

Računalništvo v oblaku postaja vse bolj razširjen pojem. Je priljubljena marketinška besedna zveza, s katero se označuje širok nabor storitev različnih produktov. Napredek spletnih tehnologij in splošna priljubljenost mobilnih naprav sta ustvarila zlato dobo računalništva v oblaku. Socialna omrežja, mobilne aplikacije ter znane spletne storitve, kot so Dropbox[10], Evernote[11] in iCloud[4], vse zagovarjajo nov način mišljenja. Naši podatki niso več vezani na posamezne fizične naprave, ampak so nam dostopni vsepovsod - so del našega osebnega oblaka. Široka priljubljenost oblakov pa ni omejena zgolj na potrošnike. Vse bolj postajajo storitve oblakov privlačne tudi podjetjem. Podjetja svoje sisteme selijo na virtualne naprave, ki so del velikih podatkovnih centrov. Tako se izognejo potrebi po zahtevnem vzdrževanju strežnikov in za strojno opremo plačujejo toliko, kot je porabijo.

Kaj točno pomeni pojem računalništvo v oblaku?

Kdo so največji igralci na tem področju?

Kako ugotoviti, koliko strojne opreme porabi neko podjetje?

Na ta in še nekatera druga vprašanja bomo skušali odgovoriti v tem diplomskem delu. Še posebej se bomo osredotočili na zadnje vprašanje, saj je merjenje aktivnosti uporabnikov eden izmed ključnih elementov vsakega sistema v oblaku.

Implementirali bomo tudi lastno spletno aplikacijo, ki bo skrbela za interpretacijo in vizualizacijo podatkov o aktivnosti uporabnikov. Spremljala bo aktivnost na trenutno najbolj popularni prosto dostopni rešitvi za ponujanje storitev v oblaku - OpenStack. Aplikacija bo sposobna obdelovanja vseh meritev, izmerjenih na oblaku. Proces obdelave bo hiter in zmožen iskanja zakonitosti v uniji več različnih tipov meritev. Vizualizacija podatkov bo pregledna in prijazna uporabnikom. Zagotovili bomo enak nivo varnosti, kot ga pozna OpenStack, z majhno kompleksnostjo procesa identifikacije za uporabnika. Za razvoj bomo uporabili tehnologije, ki so kompatibilne s platformo OpenStack, kar bo omogočalo enostavno integracijo naše aplikacije v sistem oblaka.

Poglavje 2

Pregled pojmov in tehnologij

2.1 Računalništvo v oblaku

2.1.1 Definicija

Računalništvo v oblaku je pojem, ki postaja vse bolj priljubljen. Pogosto se z njim označuje vse storitve, ki tečejo preko spleta in uporabnikom omogočajo dostop z različnih lokacij in naprav.

Predstavlja model, ki omogoča deljenje računalniških virov (računska moč, pasovna širina, hramba podatkov, storitve ipd.) preko omrežja, uporabnik pa jih lahko pridobiva in skalira brez posredovanja osebja ponudnika oblaka.[16] Vidni so mu kot strojna oprema, ki jo je mogoče dodajati in odstranjevati glede na njegove potrebe. To strojno opremo simulira programska oprema, ki zmožnosti ene ali več fizičnih naprav deli med uporabnike. Ker takšna virtualna strojna oprema fizično ne obstaja, jo je mogoče spreminjati kadarkoli, brez večjega vpliva na končnega uporabnika.

Za sisteme računalništva v oblaku veljajo lastnosti, skupne vsem načinom implementacije. Značilnosti lahko povzamemo v petih točkah.[16]

1. **Storitev na zahtevo**

   Uporabnik lahko do storitve dostopa in jo spreminja (povečevanje in zmanjševanje procesorske moči, pasovne širine, pomnilnika, količine

prostora za hrambo podatkov ...) po potrebi, brez posredovanja osebja ponudnika oblaka.

2. **Dostop preko omrežja**
   Nadzor nad oblakom je mogoč z namensko programsko opremo (odjemalci) preko spletnih brskalnikov, pametnih telefonov ali tablic.

3. **Deljenje virov**
   Fizični viri ponudnika so združeni in ponujeni uporabnikom kot bazen zmogljivosti, iz katerega preko večuporabniškega sistema uporabniki črpajo vire glede na svoje potrebe. Količina virov, ki si jih uporabnik prilasti, vpliva na ceno storitve. Končni uporabnik ponavadi nima nadzora in informacije o lokaciji fizičnih naprav, zato sistem daje občutek neodvisnosti od lokacije.

4. **Elastičnost**
   Količino fizičnih zmogljivosti lahko ponudnik enostavno in ponavadi avtomatsko poveča ali zmanjša glede na potrebe uporabnikov. Končnim uporabnikom je količina ponujenih zmogljivosti navidez neskončna.

5. **Nadzor nad porabo**
   Podatki o porabi virov se samodejno beležijo in so dostopni tako ponudniku kot uporabniku. Zagotavljajo transparentnost opravljenih storitev in omogočajo postavljanje cen glede na porabo. Tip storitve in način plačevanja za to storitev pogosto vplivata tudi na podatke, ki se beležijo (količina hrambe podatkov, procesorska moč, pasovna širina, število uporabnikov).

V diplomskem delu se ukvarjamo s peto točko, saj implementirane programske rešitve ponujajo način organizacije in vizualizacije podatkov o porabi virov v oblaku.

Sistem za ponujanje storitev računalništva v oblaku ima fizični nivo, ki ga sestavlja strojna oprema, potrebna za podporo ponujenih storitev. To so

strežniki, prostor za hrambo podatkov in omrežne komponente. Konceptualno teče nad fizičnim nivojem še nivo abstrakcije. Preko njega programska oprema, nameščena preko fizičnega nivoja, nudi dostop do virov in skrbi za njihovo deljenje. Glede na način implementacije oblačne storitve ima uporabnik nadzor nad različno velikim delom sistema. Značilni so trije modeli.[16]

1. **Programska oprema kot storitev (*Software as a Service* - SaaS)**
   Uporabniku so ponujene že obstoječe aplikacije, ki tečejo na sistemu v oblaku. Do njih lahko dostopa preko različnih odjemalcev z več naprav. Gre lahko npr. za spletno pošto, do katere pride preko brskalnika, ali pa za programski vmesnik, ki komunicira z oblakom. Vsem aplikacijam je skupno, da lahko uporabnik operira le z njimi, nima pa dostopa do infrastrukture, na kateri tečejo. Če pride do povečanih zahtev, se količina virov poveča samodejno.

2. **Platforma kot storitev (*Platform as a Service* - PaaS)**
   Model omogoča namestitev aplikacij, ki jih je pridobil ali ustvaril uporabnik sam, vendar morajo biti narejene iz nabora programskih jezikov, knjižnic, storitev in orodij, ki jih podpira ponudnik. Ponovno ni mogoč nadzor nad infrastrukturo, je pa pogosto omogočeno spreminjanje nastavitev okolja, v katerem tečejo aplikacije.

3. **Infrastruktura kot storitev (*Infrastructure as a Service* - IaaS)**
   Uporabnik si lahko dodeljuje osnovne računalniške vire (procesorska moč, pomnilnik, hramba, omrežje) in na njih poganja lastno programsko opremo. Omejitev glede uporabljenih tehnologij pri izdelavi programske opreme tu ni, saj si lahko uporabnik ustvari lastno okolje, ki podpira zahteve njegovih aplikacij. Še vedno nima nadzora nad infrastrukturo oblaka, ima pa možnost urejanja operacijskih sistemov, načina hrambe podatkov, omrežnih nastavitev in aplikacij, ki tečejo na oblaku.

Tip modela je odvisen tudi od ciljnih uporabnikov in njihovega števila. Dostop do storitev nekega oblaka je lahko namreč javen ali pa omejen na

specifično množico uporabnikov. Glede na tip namestitve lahko storitve v oblaku razvrstimo v štiri skupine.[16]

1. **Zasebni oblak**
   Oblak je namenjen uporabi znotraj posamezne organizacije. Zagotavlja ga lahko organizacija sama ali njen zunanji partner. Infrastruktura, namenjena oblaku, je lahko nameščena tudi znotraj lastnih prostorov.

2. **Oblak skupnosti**
   Uporabniki oblaka so pripadniki skupine organizacij, povezanih s skupnimi zahtevami. Lastnik in upravitelj oblaka skupnosti je ena ali več organizacij ter zunanji partner. Infrastruktura je ponovno lahko nameščena znotraj prostorov upravitelja.

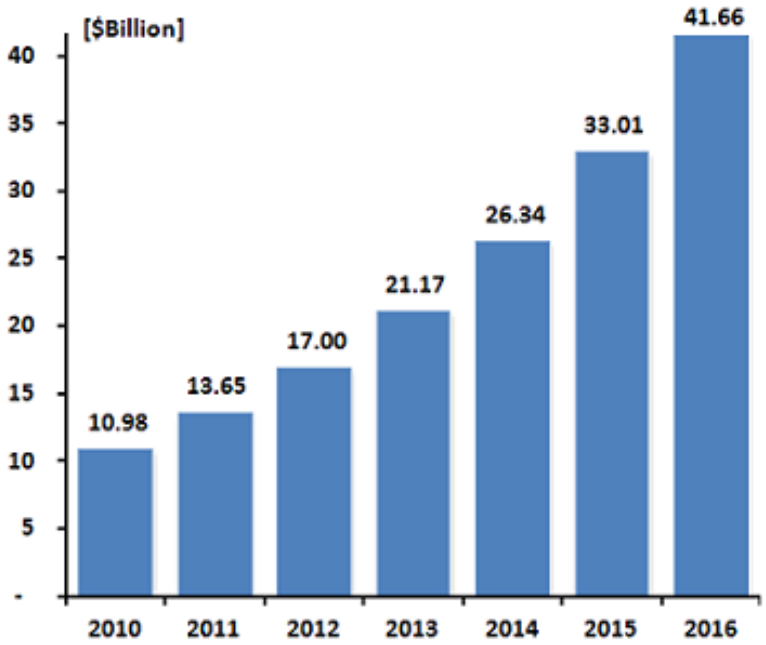
3. **Javni oblak**
   Ta tip oblaka je namenjen splošni javnosti. Zagotavljajo ga lahko podjetje, državna ali akademska organizacija, v nekaterih primerih pa tudi njihova kombinacija. Fizično je postavljen v prostorih ponudnika.

4. **Hibridni oblak**
   Hibridni oblak je sestavljen iz dveh ali več samostojnih infrastruktur iz prvih treh skupin, povezanih v celoto. Implementacije hibridnih oblakov ponavadi izkoriščajo del storitev vsakega izmed oblakov, ki jih vsebujejo. Podjetje tako npr. uporablja javen oblak za hrambo splošnih podatkov in lasten, zasebni oblak za operiranje s podatki svojih strank.

2.1.2 Ponudniki

Na trgu obstaja precejšnje število ponudnikov storitev računalništva v oblaku. Večinoma gre za komercialna podjetja, ki se ukvarjajo z zagotavljanjem IaaS in PaaS storitev. Trenutno najuspešnejši ponudniki plačljivih storitev so Amazon, Google, Microsoft, IBM, Rackspace in Salesforce. Med največjimi še vedno izrazito prednjači Amazon, saj s svojim Amazon Web Services (AWS) zaseda daleč največji tržni delež. Kljub temu ostali konkurenti trdo

delajo, da bi povečali svojo prisotnost na hitro rastočem trgu. IBM je tako pred kratkim prevzel obetavno podjetje SoftLayer, s čimer je precej razširil spekter podjetij, ki so jim njegove storitve namenjene[7], Google pa pospešeno razvija svoj prenovljen in izboljšan sistem Google Cloud Platform. Povpraševanje po IaaS/Paas storitvah narašča (slika 2.1), tako da bodo podjetja imela še mnogo priložnosti za robustne dobičke. Seznam največjih se zato lahko hitro spremeni.



Slika 2.1: Napoved rasti trga računalništva v oblaku za ZDA (v milijardah dolarjev), med leti 2010 in 2016.[15]

Zaradi rastočega zanimanja za storitve v oblaku so organizacijam vse bolj privlačne tudi implementacije lastnih, zasebnih oblakov. Cenovno ugodno alternativo skušajo takim organizacijam ponuditi številne prosto dostopne programske rešitve za implementacijo oblakov. Ena izmed njih, imenovana Eucalyptus, je znana po tem, da omogoča enostavno postavitev hibridnega oblaka. Uporablja namreč orodja in skripte, s katerimi je mogoče nadzorovati tudi AWS instance. Tako predstavlja izvrstno rešitev za postavitev razvojne

in testne platforme za aplikacije, ki bodo kasneje tekle na AWS (in obratno). Znane so tudi nekatere PaaS rešitve, kot sta OpenShift in CloudFoundry. Omogočajo hitro postavitev, kreiranje in testiranje aplikacij v okviru podprtih programskih jezikov (Ruby, Python, PHP, Perl, Java ...) in tehnologij (Git, Jenkins, Apache ...).[12] Trenutno najbolj znana odprtokodna rešitev pa je zagotovo OpenStack, ki omogoča kreiranje oblakov, ki so po funkcionalnostih konkurenčni tudi največjim komercialnim ponudnikom (Amazon, Microsoft ...). V njegovem okviru tečejo tudi programske rešitve, ustvarjene za namene tega diplomskega dela.

2.2 OpenStack

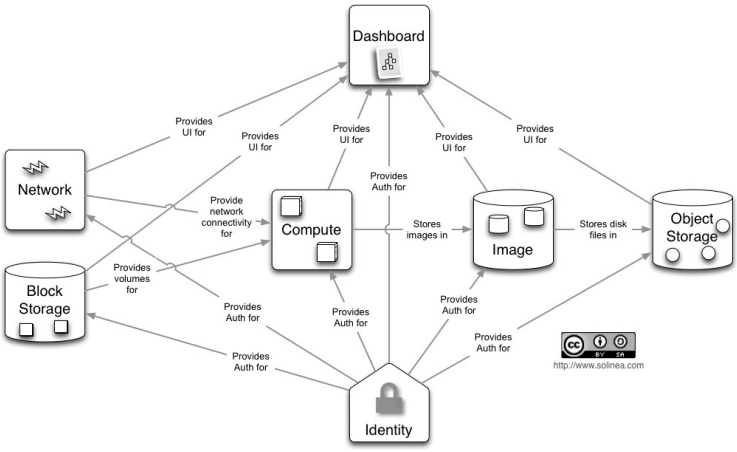
2.2.1 Zgodovina

Projekt se je začel julija 2010, ko sta Rackspace Hosting in NASA predstavila idejo o sistemu, imenovanem OpenStack, ki naj bi organizacijam pomagal pri ponujanju storitev v oblaku na standardni strojni opremi. Začetna programska koda je bila prilagojena s platform Nebula (NASA) in Rackspace Cloud Files (Rackspace Hosting). Štiri mesece kasneje je bila izdana prva uradna različica, imenovana Austin.[24] V času pisanja te naloge je aktualna osma izdaja, imenovana Havana. Razvoj poteka v ciklih, ki trajajo približno šest mesecev. Med načrtovalno fazo vsakega cikla se skupnost zbere na srečanju, na katerem potekata pregled trenutnega stanja projekta ter predstavitev in načrtovanje novih funkcionalnosti.

2.2.2 Zgradba

OpenStack je sestavljen iz več samostojnih komponent, ki, združene v celoto, nudijo celovit IaaS sistem. Vsaka ima implementiran lasten, javno dostopen API, ki jim omogoča medsebojno sodelovanje, in skrbi za prenos vhodnih in izhodnih podatkov. Ker je preko njih mogoč popoln nadzor in delovanje storitev, je vsak del OpenStack platforme mogoče precej enostavno zamen-

jati z lastno implementacijo. Edina zahteva je, da ta implementacija zna komunicirati z API-ji ostalih komponent. Takšna modularnost zagotavlja izjemno skalabilnost in enostavno nadgrajevanje. Večji del aktualnih storitev in povezav med njimi je prikazan na sliki 2.2.



Slika 2.2: OpenStack sestavlja več medsebojno sodelujočih komponent.[23]

2.2.3 Glavne komponente

Zaradi relativno enostavnega razširjanja ima OpenStack precejšnje število komponent, ki se s skoraj vsako novo izdajo še poveča. Med njimi so najpomembnejše tiste, ki zagotavljajo tri temeljne stebre računalništva v oblaku (*Compute*, *Storage*, *Networking*).[24]

1. ***Compute*** **(Nova)**
   Nova, osrednji del IaaS sistema, je prisoten že od začetne verzije in na zahtevo uporabnikov omogoča distribucijo računskih zmožnosti strojne opreme. Modul je zasnovan za enostavno horizontalno povečevanje zmogljivosti. Kompatibilen je z različnimi tehnologijami virtualizacije, s pomočjo katerih uporabnikom dodeljuje vire. Skupek virov predstavlja virtualni računalnik, imenovan instanca (*instance*). Distribuirani

viri so razvijalcem dostopni preko APIjev, sistemskim administratorjem in uporabnikom pa preko grafičnih uporabniških vmesnikov.

2. ***Storage***
   OpenStack podpira tako objektno hrambo (*Object Storage*) kot blokovno hrambo (*Block Storage*).

   (a) **Objektna hramba (Swift)**
   Pri tem tipu hrambe so podatki zapisani na več trdih diskov, razporejenih po različnih strežnikih. Namenska programska oprema zagotavlja celovitost in varnostne kopije podatkov. Horizontalno povečanje zmogljivosti je enostavno, zaradi programskega zagotavljanja varnosti podatkov pa je izvedljivo tudi z uporabo cenovno ugodnejše strojne opreme.

   (b) **Blokovna hramba (Cinder)**
   Sistem skrbi za ustvarjanje blokov in njihovo dodajanje ter odstranjevanje s strežnikov. Bloki za hrambo podatkov so integrirani v instance, ki jih ustvari Nova, in uporabniku omogočajo, da sam upravlja s hrambo svojih podatkov. S posnetki (*snapshots*) volumnov pa ponuja sistem tudi enostavno ustvarjanje varnostnih kopij, s katerimi se podatke lahko obnovi ali podvoji.

3. ***Networking*** **(Quantum, kasneje Neutron)**
   Glavne naloge komponente so nadzor in urejanje omrežij ter IP naslovov. Z uporabo plavajočih IP-jev omogoča preusmerjanje prometa v času vzdrževalnih del ali napak. Podpira tudi razširitve in tako omogoča namestitev številnih dodatnih storitev, kot so sistem zaznavanja vdorov (IDS), uravnovešanje obremenitev (*load balancing*), požarni zidovi (*firewall*) in virtualna privatna omrežja (VPN).

Omeniti je vredno še Horizon, ki predstavlja grafični vmesnik, namenjen sistemskim uporabnikom in administratorjem. Za vzdrževanje večuporabniškega sistema in identifikacijo uporabnikov skrbi Keystone. Podatke o porabi virov

preko vseh komponent pa beleži Ceilometer. Na njem in podatkih, ki jih zbere, temelji tudi večji del tega diplomskega dela.

2.3 Ceilometer

Namen projekta je zbiranje vseh podatkov, potrebnih za zaračunavanje storitev strankam, in zagotavljanje dostopne točke za sisteme nadzora porabe.[20] Podatke zbira za večino obstoječih komponent in omogoča enostavno dodajanje podpore zbiranja za nove. Uporabnikom z ustreznimi dovoljenji so podatki na voljo preko REST API.

2.3.1 Razlaga ključnih pojmov

Za beleženje podatkov Ceilometer definira lasten nabor entitet, ki predstavljajo različne elemente v strukturi podatkov. Najpomembnejše so tri temeljne entitete[26]:

Vir (*Resource*) - Eksterno definiran objekt, s katerim lahko upravljamo in za katerega se beležijo meritve. Vsaka OpenStack komponenta ima lasten nabor virov.

Števec (*Meter*) - Predstavlja kategorijo meritev. Vsak vir ima definirane svoje tipe meritev, vsak tip predstavlja en števec.

Vzorec (*Sample*) - Posamezna meritev nekega števca.

Te entitete zadostujejo za natančno beleženje aktivnosti v posameznih komponentah. Iz množice vzorcev posameznega števca je s pomočjo ustreznih funkcij mogoče pridobiti različne podatke. Osnovne agregate vzorcev (povprečje, vsota, minimum, maksimum) računa sistem tudi sam in jih ponuja v obliki statistik (*Statistics*).

Poleg zbiranja in zagotavljanja podatkov Ceilometer podpira tudi obveščanje, ko posamezne meritve prekoračijo ali padejo pod meje, ki jih je definiral

uporabnik. V takšnem primeru komponenta na določen naslov pošlje obvestilo v obliki alarma (*Alarm*).

Ceilometer pri svojem delu uporablja tudi splošno uporabljene entitete OpenStack platforme, najpomembnejše med njimi pa so:

Uporabnik (*User*) - Posamezen uporabnik sistema.

Instanca (*Instance*) - Virtualni računalnik, ki nastane z dodelitvijo določene količine potrebnih virov.

Projekt/Najemnik (*Project/Tenant*) - Ta struktura, znana pod več imeni, predstavlja projekt oz. najemnika sistema. Strukturo sestavlja skupina uporabnikov, ki znotraj njenega okvira ustvarja in uporablja instance ter ostale vire.

2.3.2 Delovanje

Glavni nalogi Ceilometra sta zbiranje in hramba podatkov o aktivnosti posameznih komponent. Ker so implementacije komponent lahko zelo različne, pozna projekt tri načine zbiranja podatkov.[25]

1. **Poslušalec na vodilu (*Bus listener agent*)**
   Za pošiljanje sporočil v OpenStacku je najboljša uporaba knjižnice Oslo, ki zagotavlja natančen in zanesljiv prenos podatkov. Pri uporabi te knjižnice se ob pošiljanju sporočil na posebnem Oslo vodilu objavljajo dogodki, ki jih poslušalec zazna, podatke pa zabeleži kot vzorce. Zaradi robustnosti takšne implementacije je to najboljša rešitev za pridobivanje podatkov. Ker pa te knjižnice marsikatera komponenta ne uporablja, poslušalec vodila Oslo ne zadostuje.

2. **Pozivni agenti (*Polling agents*)**
   Ta metoda v rednih intervalih samodejno pridobiva podatke s točk, preko katerih projekti izpostavijo svoje podatke okolici (API). Takšna tehnika je najmanj priljubljena, saj je težko zagotoviti zanesljivost in ažurnost podatkov.

3. **Potisni agenti (*Push agents*)**
   Implementacija nekaterih komponent pa podatkov o aktivnosti ne izpostavi zunanjim poslušalcem. Takšne komponente morajo vsebovati lastno logiko, ki Ceilometer obvešča o njihovi aktivnosti. Tak način pridobivanja podatkov je še vedno robusten, vendar zahteva precej dodatnega dela. Vsak element, ki ga želimo nadzorovati, potrebuje namreč lastno logiko za sporočanje podatkov.

Zbrani podatki se ponavadi beležijo v podatkovno bazo. Do nje je mogoč API dostop, ki omogoča branje in pisanje podatkov. Seveda se lahko do baze dostopa tudi neposredno, vendar to ni priporočeno. Shema in slovar podatkovne baze se z razvojem namreč spreminjata, kar lahko razvijalcem povzroča težave. Z dostopom preko API-ja se je takšnim težavam mogoče izogniti, saj ta v okviru posameznih verzij tudi ob spremembah podatkovne baze ostaja nespremenjen. Kljub temu da je Ceilometer del OpenStacka, ni omejen z njegovo definicijo uporabnikov. Z navedbo lastnega vira, iz katerega prihajajo podatki, so ti lahko zabeleženi tudi za definicije uporabnikov drugih aplikacij. Ker API dostop podpira tudi pisanje podatkov, je v podatkovno bazo mogoče zapisovati lastne podatke, specifične za aplikacijo, v okviru katere teče Ceilometer. Tako je z njim mogoče zbirati podatke za aplikacije, ki tečejo nad OpenStackom (na PaaS ali SaaS nivoju), in uporabljati enoten sistem za beleženje aktivnosti celotnega sistema oblaka.[25]

Podatki, ki jih zbere Ceilometer pa so šele začetek poti. Informacije je potrebno obdelati, med njimi najti zakonitosti in jih predstaviti uporabnikom. Teh nalog Ceilometer ne opravlja, saj se osredotoča le na zbiranje, hranjenje in ponujanje podatkov okolici. Procesiranje in vizualizacija sta prepuščena ločenim programskim rešitvam, med katere spada tudi naša aplikacija.

Poglavje 3

Definicija problema

3.1 Ekonomija računalništva v oblaku

Če želi organizacija zagotavljati stabilnost svoje spletne storitve, mora upoštevati potencialne sunke uporabe. Spletna trgovina ima lahko na primer močno povečan promet v času božičnega nakupovanja, spletna stran potovalne agencije tik pred poletnimi počitnicami, mobilna aplikacija pa ob izdaji posodobitve z novimi funkcionalnostmi. Da so strežniki sposobni prenesti takšne konice, morajo biti mnogo zmogljivejši, kot je to potrebno pri njihovem vsakdanjem delu. Tako so večino časa neizkoriščeni in predstavljajo nepotrebne stroške.

Ena glavnih prednosti računalništva v oblaku je prilagodljivost potrebam uporabnikov. Ker je vire mogoče dinamično dodajati in odstranjevati, so lahko strežniki prilagojeni trenutnim zahtevam, plačevanje za uporabo po uri pa podjetjem omogoča minimizacijo stroškov. Prav tako je mogoče doseči zelo dobro razmerje med ceno in časom izvajanja pri enkratnih, zelo zahtevnih procesiranjih (npr. analiza podatkov) - cena uporabe 1000 Amazon EC2 naprav 1 uro je enaka uporabi 1 EC2 naprave 1000 ur.[5]

Zrnato zaračunavanje storitev pa seveda ni mogoče brez natančnega nadzora nad aktivnostjo v oblaku. Tako ponudniki kot uporabniki potrebujejo jasen pregled nad porabo, saj lahko neodgovorno dodeljevanje virov hitro pripelje do visokih računov za uporabnika. Številni ponudniki nudijo celo

posebna obveščanja o nepričakovanih dvigih aktivnosti uporabnikovih instanc. Podatki o aktivnosti so koristni tudi za pridobivanje različnih statistik. Uporabniki lahko iz preteklih podatkov na primer razberejo, kdaj so njihove storitve priljubljene in kdaj niso, ter temu prilagodijo svoj poslovni model.

3.2 Zaračunavanje storitev in OpenStack

Proces zaračunavanja storitev v oblaku lahko razdelimo v tri korake[25]:

1. Merjenje (*Metering*) je proces zbiranja podatkov o tipu storitve, času, uporabniku in količini porabe. Podatki so zabeleženi v obliki vzorcev, namenjenih nadaljnjemu procesiranju.

2. Ovrednotenje (*Rating*) je proces analiziranja skupin vzorcev, združenih v enote, katerim se glede na marketinška pravila dodeli denarna vrednost.

3. Zaračunavanje (*Billing*) je združevanje cen različnih enot v račun, ki se ga lahko izda stranki.

Pri projektu OpenStack za zbiranje podatkov skrbi komponenta, imenovana Ceilometer. Podatke lahko shranjuje v datoteko, ki se jo uporabi za ločeno procesiranje. Ponavadi pa se podatki shranjujejo v podatkovno bazo, do katere je urejen dostop preko REST API.[25] Sistem zbiranja podatkov je dobro urejen in pokriva večino komponent. Dobro je podprto tudi dodajanje novih meritev in njihovo vključevanje v obstoječ sistem.

S tem pa se okvir nalog Ceilometra tudi zaključi. Namenjen je namreč le izpolnjevanju zahtev prve točke procesa zaračunavanja.[25] Takšni odločitvi botruje predvsem dejstvo, da je OpenStack prosto dostopen projekt in se zato implementacije oblakov, ustvarjene z njim, močno razlikujejo med seboj. Po eni strani gre lahko za polno komercialne namestitve, ki potrebujejo natančen in pravno zanesljiv sistem zaračunavanja, po drugi strani pa se lahko z OpenStackom ustvarijo tudi popolnoma brezplačni oblaki, namenjeni javni uporabi ali izobraževanju. Tu ne govorimo več o zaračunavanju

storitev, pač pa so podatki o aktivnosti namenjeni beleženju statistik in informiranju uporabnikov o njihovi prisotnosti v oblaku.

Zaradi izjemnega razpona namenov, ki jih takšni oblaki lahko imajo, OpenStack trenutno nima komponente, ki bi skrbela za vizualizacijo zabeleženih podatkov. Osnovne statistike je mogoče pridobiti preko nadzorne plošče namestitve, imenovane Horizon, podrobnejše prikazovanje podatkov pa je prepuščeno razvijalcem.

3.3 Cilj diplomskega dela

Ustvariti želimo spletno aplikacijo, ki bo komunicirala s Ceilometrom in pridobivala podatke preko REST API. Pridobljene podatke bo identificirala, grupirala in vizualizirala na uporabniku prijazen način. Uporabljala bo enak večuporabniški sistem kot OpenStack namestitev, s katero komunicira. To pomeni, da bo vsak uporabnik, ki ima uporabniški račun v tem sistemu, lahko uporabljal tudi našo aplikacijo. Ker imajo uporabniki različne nivoje pravic, bo to vidno tudi v aplikaciji - administrator bo imel pregled nad celotnim sistemom, običajen uporabnik pa le nad delom, v katerem je prisoten.

Glavni namen aplikacije ni zaračunavanje storitev, temveč predstavitev podatkov o aktivnosti uporabnikov. Aplikacija zato ne bo imela logike za ovrednotenje porabe, temveč bo prikazovala statistike aktivnosti različnih komponent. Omogočala bo sprehod po API-ju, na katerem bo uporabnik z izbiro poti posredno določil načine filtriranja podatkov. To ga bo v končnem koraku pripeljalo do podatkov za želeno področje. Aplikacija bo znala prebrati in prikazati podatke za vsako komponento, ki je kompatibilna s trenutno verzijo Ceilometer API-ja. Za posamezne komponente bo mogoče definirati tudi ločeno logiko vizualizacije, specifično za njihove podatke.

Za razvoj bomo uporabili enake tehnologije, kot so uporabljene za izdelavo komponente, namenjene upravljanju oblaka. Tako bo mogoče našo aplikacijo uporabljati kot samostojno orodje, ali pa jo integrirati neposredno v nadzorno ploščo sistema.

Poglavje 4

Sorodne rešitve

Pregled nad aktivnostjo je ključnega pomena za zanesljivo in ekonomično delovanje aplikacij v oblaku. Zanj obstajajo številne rešitve, med katerimi prihaja do velikih razlik, saj imajo različne glavne cilje. Namenjene so lahko informiranju uporabnikov o izkoriščenosti njihovih virov, argumentiranju končnih cen storitev, obveščanju o odstopanjih posameznih meritev (npr. izkoriščenost CPU) od definiranih norm ali kombinaciji več naštetih elementov. Razlikujejo se tudi po ceni, saj so lahko brezplačne ali pa zahtevajo mesečna plačila.

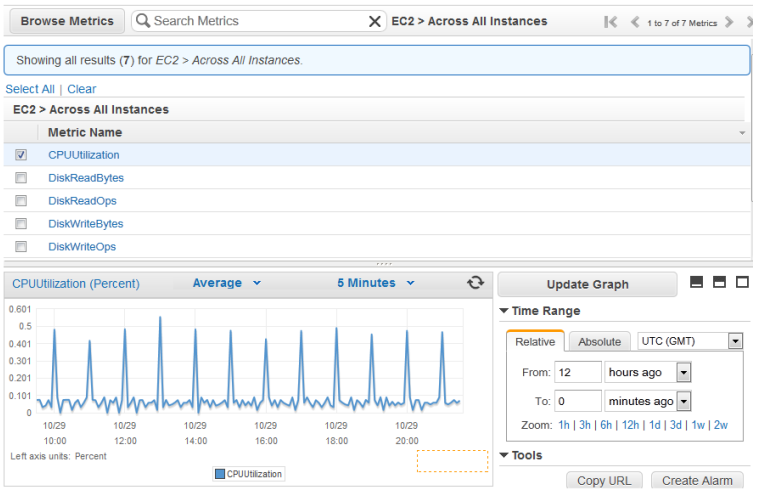
4.1 Nadzor kot storitev

Komercialni ponudniki storitev v oblaku uporabljajo lastne rešitve za nadzor nad zasedenostjo virov, ki so lahko integrirane v nadzorne plošče ali pa jih nudijo kot eno izmed svojih storitev. Implementacije večjih ponudnikov so si zaradi sorodnih namenov precej podobne, še vedno pa med njimi prihaja do razlik.

4.1.1 Amazon CloudWatch

Amazon CloudWatch uporabnikom nudi pregled nad viri in delovanjem aplikacij v AWS. Zbira različne metrike, specifične za posamezne komponente v

sistemu. Uporabnik lahko na primer pri EC2 instancah spremlja izkoriščenost CPU (slika 4.1), prenos podatkov in aktivnost uporabe diskov. Pri storitvi čakalne vrste (SQS) lahko spremlja število poslanih in prejetih sporočil, pri storitvi obveščanja (SNS) pa število objavljenih in dostavljenih sporočil.[2] Z uporabo API klicev lahko razvijalci zbirajo tudi metrike, ki jih ustvarjajo njihove aplikacije na AWS. Zbrani podatki so uporabniku vidni preko nadzorne konzole.

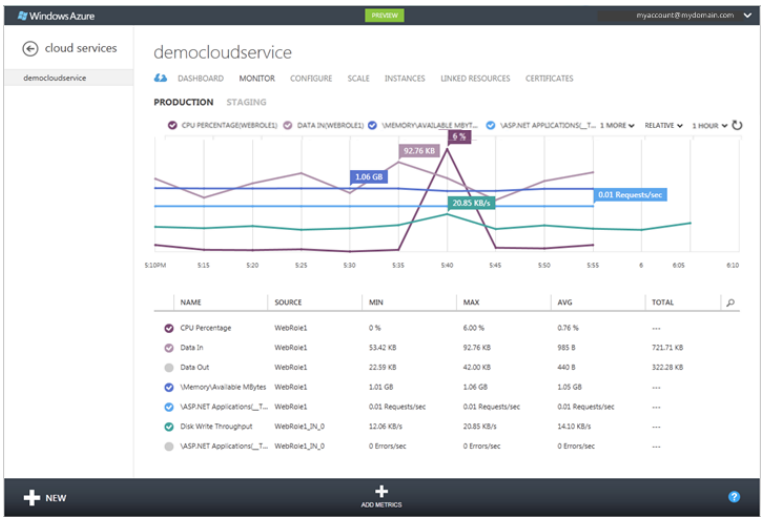


Slika 4.1: Vizualizacija izkoriščenosti CPU za EC2 instanco preko nadzorne konzole.[1]

Do podatkov je omogočen tudi programski dostop preko API. Uporaba te storitve pri velikem številu zahtev postane plačljiva. Uporabnik lahko za želene metrike ustvari tudi alarme, ki pošiljajo obvestila, ko metrike zapustijo določene meje. CloudWatch je na voljo v brezplačni različici (*Free Tier*), ki vključuje osnovne metrike s petminutnim intervalom zbiranja podatkov. Stranke so upravičene še do desetih dodatnih metrik po svojem izboru, desetih alarmov in milijona API zahtev na mesec.[3] Izboljšanje storitev (zbiranje podatkov z manjšim intervalom, dodatne metrike in alarmi) zahteva mesečna plačila, ki se določajo glede na količino dodatnih storitev.

4.1.2 Windows Azure

Pri Windows Azure je sistem za pregled nad aktivnostjo sestavni del nadzorne plošče in ni tržen kot ločena komponenta. Kljub temu je zelo podroben in omogoča spremljanje podatkov za storitve v oblaku (izkoriščenost CPU, vhodni in izhodni podatki, branje in pisanje na disk)[18], za storitve hrambe podatkov (razpoložljivost, latenca, odstotek uspešnosti)[17] in celo za spletne strani, ki tečejo na tem sistemu (uporaba CPU, količina prejetih in poslanih podatkov, tipi in število uspešnih in neuspešnih zahtev)[19]. Kot je vidno na sliki 4.2, je na grafu mogoče hkrati prikazati krivulje različnih meritev.



Slika 4.2: Vizualizacija izbranih metrik na nadzorni konzoli Windows Azure.[18]

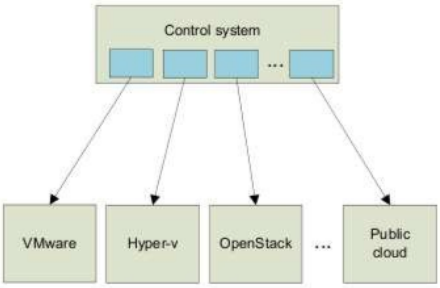
4.2 Hibridni oblaki

Ker so hibridni oblaki sestavljeni iz delov različnih infrastruktur, je pri njih težko ustvariti enoten sistem nadzora. Do težav prihaja predvsem pri pridobivanju podatkov, saj vsaka infrastruktura uporablja lastne entitete in koncepte povezav med podatki. Podatki so ponavadi programsko dostopni,

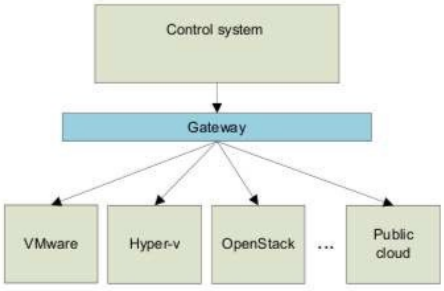
vendar vsaka storitev ponuja lasten API, zato vsaka komponenta hibridnega oblaka zahteva dodatno logiko za pridobivanje podatkov. Koncept reševanja takšnih problemov in testna implementacija univerzalnega sistema za nadzor nad aktivnostjo v treh različnih oblakih (VMware, HyperV in OpenStack) sta predstavljena v [28].

Infrastruktura avtorjev članka zbira podatke in je sposobna komunikacije z različnimi sistemi oblakov. Zbrane podatke vizualizira v nadzornem sistemu, kjer so na voljo uporabniku. Takšen koncept omogoča nadzor nad poljubnim številom oblakov, potrebna je le logika za komunikacijo, sposobna sporazumevanja z vsemi sistemi. Za zbiranje podatkov z različnih platform avtorji predstavijo dva potencialna tipa arhitektur.

Prvi tip predvideva neposredno komunikacijo nadzornega sistema z vmesniki virtualnih platform (slika 4.3). To zahteva namestitev dodatnih modulov v kontrolni sistem za vsak vmesnik. Glavni prednosti takšne implementacije sta lažje razhroščevanje in boljši nadzor nad napakami. Pri drugem načinu implementacije pa komunikacija poteka posredno, preko prevajalnega vmesnika (slika 4.4). Vmesnik skrbi za komunikacijo s posameznimi virtualnimi platformami, nadzorni sistem pa komunicira z vmesnikom po ločenem protokolu. Vsi nadzorovani elementi so tako predstavljeni na enak način, zato je sistem lažje razširjati. Takšna arhitektura ne zahteva namestitve dodatnih modulov v nadzorni sistem, je pa potrebna implementacija prevajalnega vmesnika.



Slika 4.3: Neposredna komunikacija nadzornega sistema.[28]



Slika 4.4: Uporaba prevajalnega vmesnika.[28]

4.3 OpenStack

Za zbiranje podatkov na OpenStack platformi obstaja več sistemov.[13]

- Zabbix je prosto dostopna in splošno namenska rešitev zbiranja podatkov, ki jo je mogoče prilagoditi za uporabo z OpenStackom.

- Synaps je sistem za nadzor, kompatibilen z AWS CloudWatch komponento. Zadolžen je za zbiranje podatkov, zagotavljanje statistik in obveščanje uporabnikov glede na ustvarjene alarme.

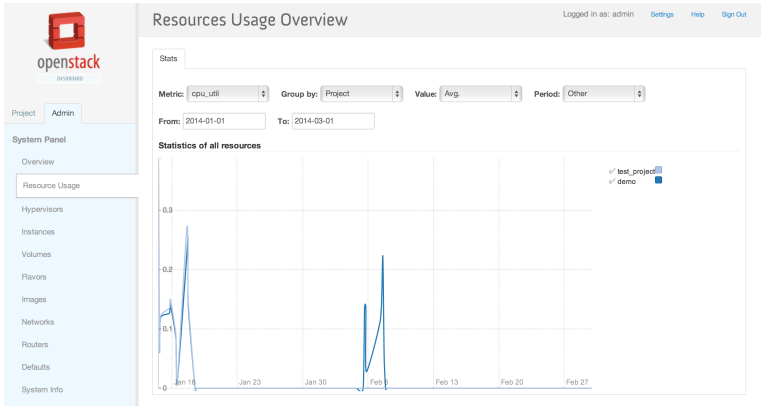
- Healthnmon ponuja podjetje Hewlett-Packard kot storitev za nadzor nad viri in aplikacijami v oblaku.

- StackTach je namenjen nadzoru in razhroščevanju. Sposoben je dela z več podatkovnimi centri hkrati.

Najbolj znan, obetaven in tudi podprt kot uradna komponenta projekta OpenStack pa je že opisani Ceilometer. Meritve Ceilometra deloma vizualizira OpenStack uporabniški vmesnik Horizon (slika 4.5), vendar so njegove funkcionalnosti trenutno še precej omejene. Vizualizacija je na voljo le administratorjem in nudi samo globalni pregled porabe virov v posameznih projektih.

4.3.1 Talligent

Projekt Talligent se trenutno oglašuje kot »prva robustna rešitev zaračunavanja za OpenStack«.[27] Zasnovan je kot samostojna aplikacija z lastnim večuporabniškim sistemom in naborom entitet. Ob namestitvi je potrebno določiti dostopne točke do OpenStack sistema in ustvariti sistem uporabnikov. Posameznim elementom, katerih uporabo aplikacija nadzoruje, je nato mogoče določiti cenovne vrednosti, na podlagi katerih aplikacija ustvarja končne račune. Vmesnik za določanje vrednosti je prikazan na sliki 4.6.

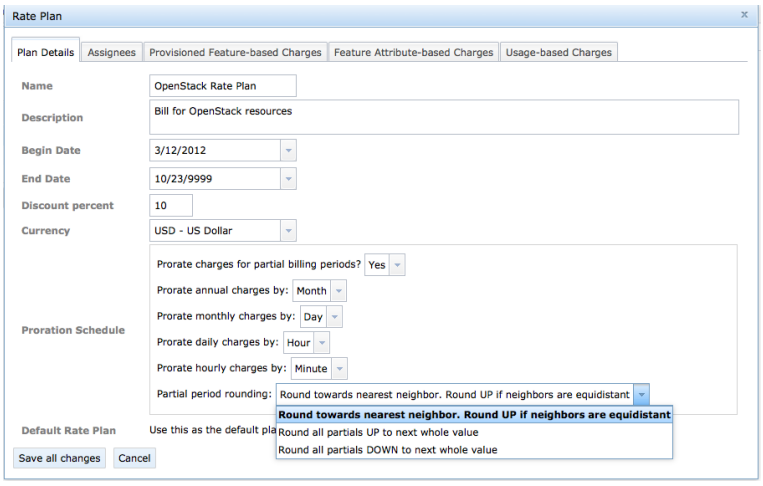


Slika 4.5: Pregled izkoriščenosti CPU v projektih preko nadzorne plošče Horizon.

4.4 Potencialne izboljšave

Na področju obdelovanja in prikazovanja podatkov o aktivnosti sistemov v oblaku je še vedno prostor za izboljšave. Največ pomanjkljivosti je moč najti ravno pri prosto dostopnih sistemih, kot je OpenStack. Komercialni ponudniki, ki sicer ponujajo kompleksna in zmogljiva orodja, so osredotočeni predvsem na nadzorovanje lastnih oblakov. Zunanja uporaba njihovih rešitev je omogočena znotraj (pogosto omejujočih) okvirov komunikacije preko API.

Presenetljivo je, da kljub splošni priljubljenosti platforme OpenStack in uradni podprtosti komponente Ceilometer zelo težko najdemo rešitve, ki uporabljajo ponujene podatke. Redke aplikacije, ki to počnejo (Talligent) pa so osredotočene na ustvarjanje računov in komercialno ponujanje storitev v oblaku. Velika prednost in potencial prosto dostopnih platform je ravno njihova uporaba v javne in izobraževalne namene. Orodja, namenjenega pregledovanju podatkov na brezplačnem, javno dostopnem oblaku trenutno praktično ni mogoče najti. Izpolnjevanje potreb te niše je eden izmed poglavitnih ciljev, ki jih želi doseči naša aplikacija.



Slika 4.6: Konfiguriranje končnega računa z aplikacijo Talligent.[27]

Poglavje 5

Implementacija aplikacije

Razvili smo spletno aplikacijo, imenovano CeiloReader. Uporabnikom služi kot orodje za komunikacijo s poljubnim oblakom, ustvarjenim z OpenStack platformo. Aplikacija uporabnikom predstavi podatke o njihovi aktivnosti v oblaku, ki jih pridobi od komponente Ceilometer.

5.1 Uporabljeni programski jeziki in tehnologije

Jedro aplikacije je spisano s pomočjo ogrodja Django[9], ki je namenjeno ustvarjanju robustnih spletnih aplikacij. Django je skupek knjižnic za delo s programskim jezikom Python. Ker je s tem ogrodjem ustvarjena tudi OpenStackova nadzorna plošča Horizon, je njena razširitev z našo aplikacijo dokaj enostavna. Tako imenovani »*front-end*« aplikacije je ustvarjen z ustaljenimi spletnimi tehnologijami HTML, CSS in JavaScript. Za vizualno urejanje uporabljamo skupek CSS in JavaScript knjižnic, imenovan Bootstrap[6]. Vizualna podoba aplikacije je precej osnovna, saj smo se primarno posvetili predstavitvi konceptov funkcionalnosti. Aplikacija za implementacijo nekaterih elementov uporablja še druge zunanje knjižnice, ki so navedene med pregledom implementacije in opisom namestitve.

5.1.1 Ogrodje Django

Django je ogrodje, namenjeno razvoju spletnih aplikacij. Njegova izvorna koda je napisana v programskem jeziku Python. Teče na strežniku Apache z nameščenim modulom *mod_wsgi*.[8] Vzdržuje seznam regularnih izrazov URL naslovov, vsak izmed njih pa predstavlja ločen pogled. Posamezni pogledi so funkcije, ki se izvedejo, ko uporabnik obišče njihov naslov. Vrnejo HTTP odgovor, ki lahko vsebuje tudi HTML predlogo za vizualizacijo spletne strani.

Ogrodje nudi pomagala za implementacijo različnih funkcionalnosti sodobnih spletnih aplikacij. Uporabili smo sisteme za overitev uporabnikov, vzdrževanje seje, izdelavo in delovanje form in druge. Django daje velik poudarek tudi delu s podatkovno bazo, ki pa smo se mu v naši aplikaciji izognili, saj komuniciramo s podatkovno bazo Ceilometra.

5.2 Pregled funkcionalnosti

Ob prihodu na začetno stran aplikacije CeiloReader se mora uporabnik vpisati. Za vpis uporabi isto uporabniško ime in geslo, kot za vpis v nadzorno ploščo Horizon. Glede na njegove pravice mu bodo znotraj naše aplikacije vidni tudi ustrezni podatki. Kot že rečeno, aplikacija uporabniku omogoča sprehod po podatkih, ki jih nudi Ceilometer. Pogosto lahko pridemo do enakih podatkov po več različnih poteh, saj vsak korak pomeni dodatno filtriranje, to pa lahko privede do enake končne množice podatkov iz več smeri. Pot po podatkih je sestavljena iz treh nivojev. Vsak nivo predstavlja podrobnejšo predstavitev podatkov izbrane komponente prejšnjega nivoja.

Na vrhu vsakega pogleda znotraj aplikacije je vidna orodna vrstica, kjer je na voljo izbira začetka pregledovanja. Naveden je tudi trenutno vpisan uporabnik z možnostjo izpisa iz aplikacije. Pregledovanje je mogoče začeti iz dveh začetnih točk, imenovanih *Meters* in *Resources*. Zavihek Števci (*Meters*) je namenjen pregledovanju podatkov posameznih števcev, zavihek Viri (*Resources*) pa pregledovanju vseh števcev za posamezen vir. Obe poti bomo podrobneje predstavili s testnim uporabnikom, ki bo imel v OpenStack okolju

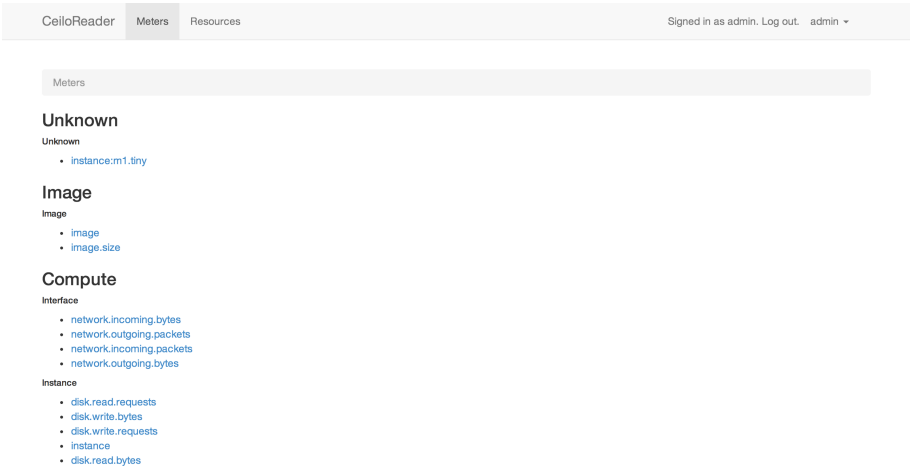
administratorske pravice (vidni mu bodo skoraj vsi podatki). Omeniti je potrebno tudi, da aplikacija predpostavlja, da je uporabnik seznanjen z meritvami Ceilometra ter njihovimi pomeni.

5.2.1 Zavihek *Meters*

Začetek filtriranja s pregledom števcev uporabimo, kadar nas zanimajo podatki o posamezni meritvi za več različnih virov. Primer uporabe je pregledovanje izkoriščenosti CPU različnih instanc.

Prvi nivo - izbira števca

Ob izbiri zavihka *Meters* so na voljo števci, za katere obstajajo meritve. Združeni so glede na komponente, ki jim pripadajo, in vire, za katere merijo podatke (slika 5.1). Uvrščanje števcev je trenutno edini proces, ki zahteva vnaprej zabeležene podatke. Ti podatki definirajo skupine, ki jim števci pripadajo. Če za katere števce še ni vnaprej zabeleženih podatkov, se združijo v ločeno skupino. Uvrščanje je podrobneje opisano v pregledu implementacije.



Slika 5.1: Prikaz števcev, za katere so zabeležene meritve. V skupini *Unknown* so števci, za katere aplikacija nima podatkov o pripadnosti.

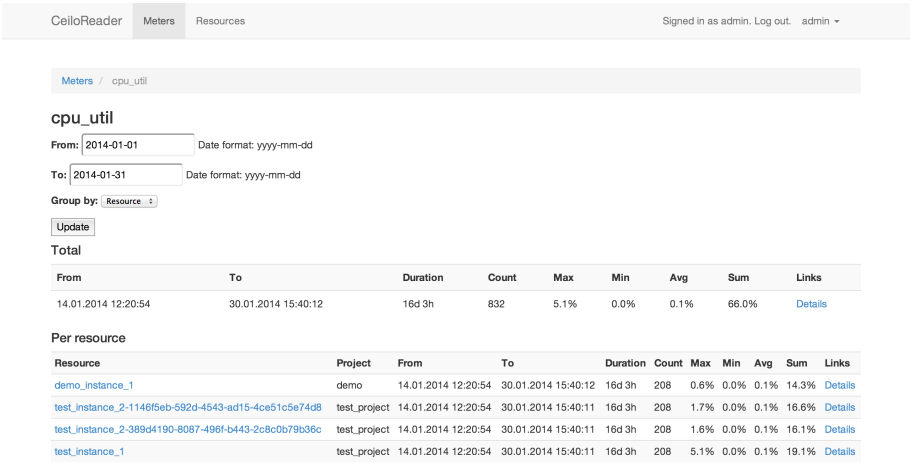
Drugi nivo - pregled statistik

Ko uporabnik izbere števec, ga aplikacija preusmeri na nov pogled, kjer je prikazan pregled statistik meritev. Statistika je posebna entiteta, ki jo ponuja Ceilometer in predstavlja povzetek množice vzorcev. To množico določimo z nastavitvijo filtrov, ki nas na tisti točki zanimajo. Statistika vsebuje podatke o največji in najmanjši izmerjeni vrednosti v naboru vzorcev, njihovo povprečje ter vsoto. Vsebuje tudi metapodatke, kot so čas prvega in zadnjega vzorca ter število vzorcev v množici.

Aplikacija omogoča omejevanje časovnega intervala prikazanih statistik in različne načine grupiranja. V pregledu je navedena splošna statistika za izbran števec ter posamezne statistike, razdeljene glede na izbran način grupiranja. Uporabnik lahko prikazane statistike grupira po virih, po projektih in, v primeru administratorja, po uporabnikih. Glede na izbrane časovne omejitve in način združevanja aplikacija nastavi ustrezne filtre ter s tem omeji množico vzorcev, iz katere Ceilometer izračuna statistike. V primeru števca, ki meri izkoriščenost CPU (*cpu_util*), združevanje po virih ustvari seznam, pri katerem vsaka vrstica predstavlja instanco, podatki pa njeno izkoriščenost CPU. Združevanje po projektih povzroči, da vsaka vrstica predstavlja posamezen projekt, podatki pa splošno CPU izkoriščenost instanc v tem projektu. Pri združevanju po uporabnikih pa statistike predstavljajo izkoriščenost CPU instanc, pri katerih posamezni uporabniki sodelujejo.

Na sliki 5.2 so prikazane statistike za izkoriščenost CPU od 1. do 31. januarja 2014, združene po virih. Vidimo, da je pri vseh statistikah prvi vzorec znotraj tega časovnega intervala zabeležen 14. januarja, zadnji pa 30. januarja. To pomeni, da so prikazani podatki za 16 dni in 3 ure, vsebujejo pa 208 vzorcev. Ker smo na teh testnih instancah le občasno izvedli enostavne funkcije, je izkoriščenost CPU zelo nizka (v povprečju 0,1%).

Pri posamezni statistiki je s klikom na povezavo *Details* mogoče videti tudi njene podrobnosti, kar uporabnika privede do zadnjega nivoja filtriranja.



Slika 5.2: Pregled statistik izkoriščenosti CPU v januarju 2014 za posamezne instance (združevanje po virih).

Tretji nivo - podrobnosti posameznih statistik

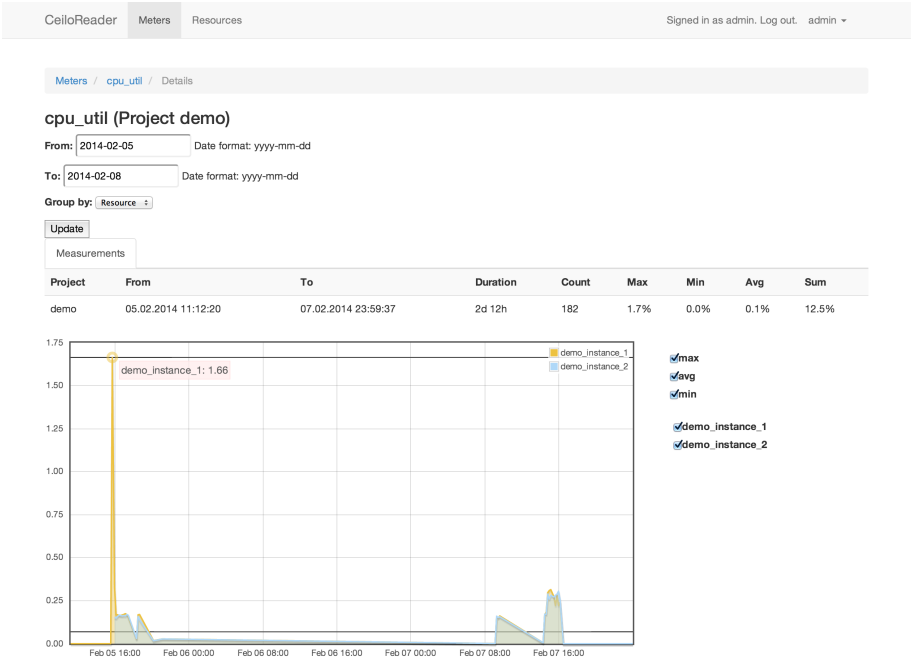
Povezava *Details* uporabnika preusmeri na tretji pogled, kjer je podrobneje predstavljena množica vzorcev, pridobljena s filtri na drugem nivoju. Ponovno je mogoče omejevanje časovnega intervala in izbiranje načina grupiranja podatkov. Tu se iz podatkov izriše graf, pri katerem vsaka krivulja predstavlja svoj nabor podatkov, odvisen od izbranega načina združevanja.

Načini združevanja in nekatere druge funkcionalnosti so na tem nivoju odvisne od prejšnjih izbir. Način grupiranja definira pomen posamezne krivulje na grafu. Če je uporabnik na prejšnjem nivoju pregledoval statistike, združene po projektih, to pomeni, da sedaj pregleduje podatke posameznega projekta. Tako ima na izbiro le grupiranje po uporabnikih in virih, krivulje na grafu pa bodo prikazovale aktivnost uporabnikov oziroma virov znotraj tega projekta.

Splošna statistika in graf so prikazani v zavihku *Measurements*. Če na tem nivoju uporabnik pregleduje podatke o viru (torej je na prejšnjem nivoju statistike združeval po virih), ima v zavihku *Info* prikazane tudi metapodatke o izbranem viru.

Krivulje na grafu so označene v legendi, v kateri je zapisano, katero entiteto predstavlja posamezna krivulja. Za boljšo predstavo o aktivnosti entitet so na grafu začrtane tudi največja, najmanjša in povprečna vrednost (ki jih je izračunal Ceilometer pri pridobivanju statistik). Te vrednosti in tudi posamezne krivulje je mogoče enostavno dodajati in odstranjevati z grafa s klikom na ustrezno vrednost v legendi. Če želi uporabnik ugotoviti natančen podatek za del grafa, se lahko s kazalcem pomakne na točko na grafu, in prikaže se mu numerična vrednost tiste točke.

Slika 5.3 prikazuje pregled izkoriščenosti CPU v projektu *demo* od 5. do 8. februarja 2014. Izbrano je združevanje po virih, posamezne krivulje torej predstavljajo aktivnost instanc v tem projektu. Vidimo, da je največji izkoristek CPU dosegla instanca *demo_instance_1*, ta pa je bil še vedno zelo nizek (1,66%).



Slika 5.3: Izkoriščenost CPU posameznih instanc v projektu demo.

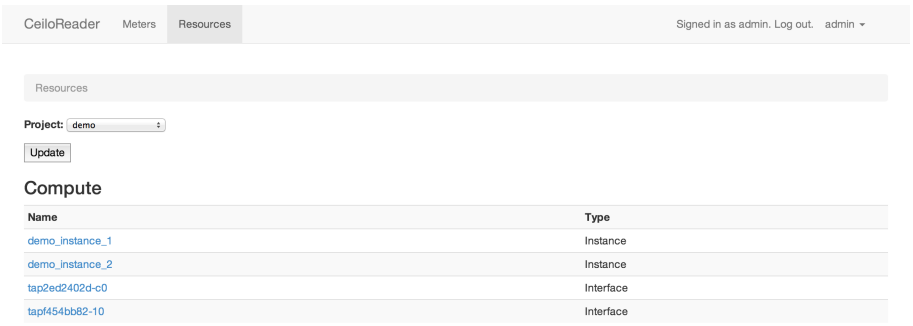
5.2.2 Zavihek *Resources*

Kadar nas zanimajo podatki o različnih meritvah za specifičen vir, filtriranje začnemo z zavihkom viri. Primer uporabe filtriranja bo prikazan s pregledom aktivnosti vira v *Compute* komponenti - instance.

Prvi nivo - izbira vira

Na prvem nivoju uporabnik najprej določi projekt, ki ga zanima. Predstavljeni so mu vsi viri, ki v izbranem projektu obstajajo. Navedeni so v seznamu, v katerem je definiran tudi njihov tip. Podobno kot števci v prvem zavihku so tudi viri združeni glede na pripadnost komponenti.

Na sliki 5.4 so navedeni viri v projektu *demo*. Projekt vsebuje le vire komponente *Compute*. Pozna dva tipa teh virov: virtualni računalnik (*instance*) in mrežni vmesnik (*interface*).



Slika 5.4: Prikaz virov v projektu *demo*.

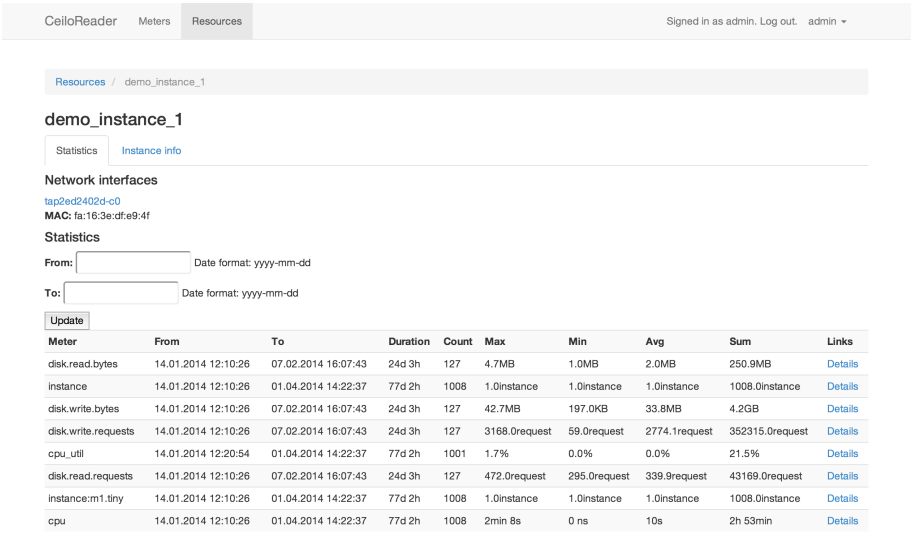
Drugi nivo - pregled statistik za posamezen vir

Na drugem nivoju je predstavljen pregled statistik vseh števcev, ki merijo podatke za izbran vir. Ponovno je mogoče omejevanje časovnega intervala, znotraj katerega Ceilometer računa statistike. Statistike so predstavljene v seznamu, s klikom na *Details* pa je mogoče videti še več podrobnosti o množici podatkov, iz katere so bile izračunane. Poleg statistik, vidnih v

zavihku *Statistics*, so prikazani tudi metapodatki izbranega vira. Dostopni so pod zavihkom *Resource info*.

Aplikacija podpira tudi definiranje ločenih pogledov za specifične vire. Nekateri viri so namreč tesno povezani z drugimi, ali pa za njih veljajo specifična pravila. Njihove podatke je tako mogoče realno prikazati le z dodatno logiko, ki ni vsebovana v splošnem načinu prikaza podatkov.

Primer uporabe ločenih pogledov je implementiran pri instancah in mrežnih vmesnikih (slika 5.5). Števci, povezani neposredno z instanco, merijo le aktivnost procesorja in operacije komuniciranja z diskom. Vsaka instanca pa je lahko povezana z enim ali več mrežnimi vmesniki. Mrežni vmesnik je ločen vir, ki nadzoruje omrežno komunikacijo instance. Tako je količina prenesenih podatkov določene instance zabeležena v njenem mrežnem vmesniku. Da so pri pregledu instanc vidni vsi podatki, imajo pogledi podatkov o instancah prikazane tudi povezave do vseh mrežnih vmesnikov, ki merijo njihovo mrežno aktivnost. Izboljšan imajo tudi prikaz metapodatkov, saj so v prikazu navedeni le za uporabnika relevantni podatki.



Slika 5.5: Pregled statistik za instanco *demo_instance_1*. Vidna je tudi povezava do mrežnega vmesnika *tap2ed2402d-c0*.

Tretji nivo - podrobnosti posameznih statistik

Pri tretjem nivoju pride do izraza dejstvo, da lahko do enakega rezultata pridemo po več različnih poteh. Preglede na tretjem nivoju je namreč z ustrezno izbiro filtrov mogoče doseči tudi v zavihku *Meters*. Vse povezave do tretjega nivoja uporabnika preusmerijo na ekvivalentno filtrirano množico podatkov v prvem zavihku (*Meters*).

Če pri *demo_instance_1* uporabnik želi videti podrobnosti o izkoriščenosti CPU, lahko pride do istega rezultata na zavihku *Meters*. Izbrati mora števec *cpu_util* in na drugem nivoju združiti podatke po virih. Nato izbere podrobnosti za *demo_instance_1*.

5.3 Način uporabe

Naša aplikacija se razlikuje od klasičnih nadzornih plošč predvsem zaradi dejstva, da uporabniku niso takoj vidni osnovni podatki, ki bi ga lahko zanimali. Takšen uvodni pregled je seveda mogoče implementirati, vendar smo se mu pri razvoju izognili.

Namen aplikacije je namreč odgovarjanje na vprašanja, ki jih postavlja uporabnik. Če ga na primer zanima, kako je njegovo delo zasedlo procesorje instanc, na katerih je delal, bo lahko hitro prišel do želenih podatkov. Ker ga zanima izkoriščenost CPU, bo izbral zavihek *Meters* in se odločil za števec *cpu_util*. Z izbiro grupiranja po virih bo že videl pregled statistik za vse vire, do katerih ima dostop. Če ga zanimajo podrobnosti za posamezen vir, so mu te dostopne s klikom na *Details*, če pa ga zanimajo ostale meritve pri tem viru, ima prav tako na voljo povezavo do pregleda meritev na zavihku *Resources*.

Na podoben način je mogoče dobiti odgovore na številna vprašanja glede aktivnosti uporabnikov v oblaku. Kljub temu da uporabnik ob vpisu v aplikacijo nima neposredno prikazanih podatkov, lahko pride do specifičnih podatkov, ki ga zanimajo, v dveh, največ treh klikih.

5.4 Pregled implementacije

5.4.1 Komunikacija z OpenStackom

CeiloReader komunicira z OpenStack namestitvijo preko REST API-jev posameznih komponent. Večino podatkov pridobi od Ceilometra, za informacije o uporabnikih in projektih pa občasno komunicira tudi s komponento Keystone. Pri obeh za komunikacijo uporabljamo uradno podprti knjižnici Ceilometer Client in Keystone Client. Keystone Client je že uporabljen in vsebovan v knjižnici django_openstack_auth, zato ga nismo ločeno nameščali.

Nekatere komponente poznajo več verzij API-jev, ki se razlikujejo po strukturi in deloma tudi vsebini. Večina uradnih klientov zna komunicirati z vsemi verzijami, ki so na voljo. Ceilometer pozna dve verziji, naša aplikacija pa komunicira z drugo, ki je novejša in bolje strukturirana. Sprva smo uporabljali tudi lastno implementacijo HTTP komunikacije (izvorna koda 1), saj na začetku razvoja Ceilometer Client še ni bil uradno podprt in ni zmogel pridobivati vseh podatkov, ki jih je nudil Ceilometer. Med razvojem je močno napredoval in nam omogočil odstranitev lastne logike za komunikacijo.

Oba klienta delujeta zelo podobno in nudita enostavno pridobivanje podatkov preko klicev ustreznih metod. Vrneta nam programske objekte, ekvivalentne entitetam, ki jih pridobivamo. V izvorni kodi 2 je prikazana uporaba klienta za pridobivanje seznama virov. Klient je ločen objekt, zadolžen za komunikacijo. Ustvarimo ga tako, da izberemo verzijo API-ja za komunikacijo, mu podamo dostopno točko in uporabnikov identifikator (*token*), potreben za verifikacijo. Seznam virov pridobimo s klicem ustrezne metode, ki ji lahko podamo seznam zahtev za filtriranje. Pridobivanje seznama zahtev za filtriranje je prikazano v izvorni kodi 4. Ob uspešni komunikaciji nam klient vrne seznam objektov tipa *Resource*, ki že vsebujejo pridobljene podatke in so pripravljeni za nadaljnje delo.

```
def fetch_data(token, path, query=''):
    """
    Fetch.
    """

    headers = {'X-Auth-Token': token, 'Content-Type': 'application/json'}
    conn = httplib.HTTPConnection(HOSTNAME + ':8777')
    conn.request('GET', '/v2/' + path, headers=headers, body=query)

    r1 = conn.getresponse()
    #status = r1.status
    #reason = r1.reason

    data = r1.read()
    conn.close()

    try:
        return json.loads(data)
    except ValueError:
        return {'error': data}
```

Izvorna koda 1: Deloma spremenjena logika za komunikacijo, ki se v trenutni obliki uporablja le v razvojne namene.

```
ceilometer = Client('2', endpoint=CEILOMETER_ENDPOINT, token=(lambda: token_id))
resources = ceilometer.resources.list(q=q)
```

Izvorna koda 2: Uporaba klienta za pridobivanje filtriranega seznama virov.

Ustvarjanje seznama zahtev za filtriranje

Filtriranje virov glede na pot, ki jo uporabnik izbere, je ključna funkcionalnost naše aplikacije. Večino filtriranja se izvede preko seznamov zahtev za filtriranje, ki jih podamo v Ceilometer Client. Ta seznam klient doda v HTTP zahtevo, oblikovano glede na zahteve API-ja. Filtri se zapišejo kot JSON ali XML seznam, v katerem je vsak element poseben slovar, ki definira zahtevo

za filtriranje. V izvorni kodi 3 je prikazan primer strukture posamezne zahteve.

V naši aplikaciji smo definirali poseben razred, imenovan *QueryFactory*, ki nam poenostavi proces ustvarjanja seznamov zahtev. Njegova uporaba je prikazana v izvorni kodi 4. Objekt, ki ga ustvarimo, vzdržuje seznam zahtev, kateremu lahko dodajamo nove zahteve. Kadar želimo, lahko pridobimo trenuten seznam zahtev, strukturiran glede na zahteve klienta.

```
{
    "field": "resource_id",
    "op": "eq",
    "type": "string",
    "value": "bd9431c1-8d69-4ad3-803a-8d4a6b89fd36"
}
```

Izvorna koda 3: Primer JSON zahteve za filtriranje[26].

```
qFactory = QueryFactory()
qFactory.add_condition(field='project_id', operator='eq', value=project_id)
q = qFactory.get_query()
```

Izvorna koda 4: Ustvarjanje seznamov zahtev za filtriranje z uporabo *QueryFactory*.

5.4.2 Overjanje in sistem uporabnikov

Želimo, da lahko CeiloReader uporablja vsak uporabnik oblaka, ki ga aplikacija nadzoruje. Da je to mogoče, je potrebno uporabnika ob vpisu overiti tako v naši aplikaciji kot na OpenStack oblaku. V aplikaciji je uporabnika potrebno verificirati s klicem posebne funkcije ogrodja Django, OpenStack pa za identifikacijo uporabnikov uporablja ločeno komponento Keystone, s katero komunicira preko REST API. Ob uspešni overitvi uporabnika Key-

stone vrne poseben identifikator, ki ga je potrebno dodati v glavo vsake HTTP zahteve ob komunikaciji s poljubno OpenStack komponento. Za ta namen uporabljamo knjižnico *django_openstack_auth*, ki uporabnika verificira v obeh sistemih in nam vrne poseben objekt *User*, ki predstavlja uporabnika v naši aplikaciji, hkrati pa vsebuje Keystone identifikator, imenovan *token*. S tem identifikatorjem se uporabnik predstavi v vsakem API klicu, kar ciljni OpenStack komponenti omogoči, da aplikaciji vrne ustrezne podatke glede na uporabnikovo pozicijo v hierarhiji uporabnikov. V aplikaciji se uporabniku ob vpisu ustvari varna seja, brez katere ni mogoč dostop do ostalih pogledov. Po končanem delu se uporabnik lahko izpiše, s čimer zaključi sejo in deaktivira svoj Keystone identifikator.

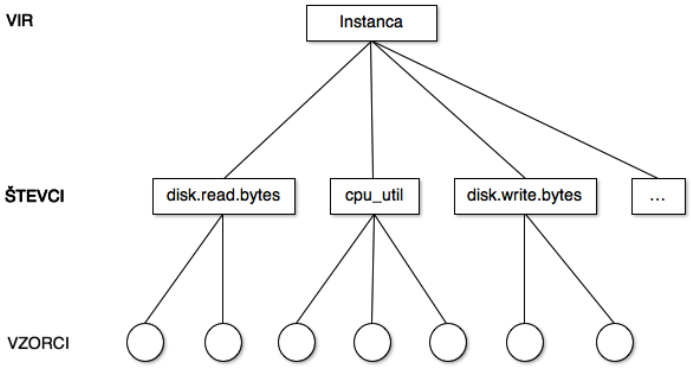
5.4.3 Interpretacija pridobljenih podatkov

Ceilometer skrbi le za zbiranje in beleženje podatkov. Do njih nudi dostop in na njih izvaja osnovne operacije (npr. računanje statistik). Interpretacija in uporaba precej kompleksne množice podatkov je prepuščena razvijalcem.

Razred *Explorer*

Za temelj interpretiranja podatkov v naši aplikaciji smo ustvarili razred *Explorer*. Ta razred izpolnjuje širok nabor nalog in poleg interpretiranja podatkov deloma skrbi še za njihovo vizualizacijo in nekatere druge specifične naloge. Z njim raziskujemo podatke, ki jih ponuja Ceilometer, kar se odraža tudi v razvojnem imenu. Osnovan je na dejstvu, da posamezen vir spada v točno določeno komponento OpenStack platforme. Vsak vir ima tudi lasten nabor števcev, ki merijo njegovo aktivnost in ustvarjajo ločeno množico vzorcev. Vsak vir je vrh hierarhije podatkov o njegovi aktivnosti (slika 5.6).

Posamezen objekt *Explorer* zato predstavlja posamezen vir. Ko pridobimo podatke o števcih, njihovih meritvah in statistikah za nek vir, jih lahko dodamo v ustrezen *Explorer*, ki jih bo zabeležil in ugotovil nekatere povezave med njimi.



Slika 5.6: Hierarhija vira. Vsak vir ima lastne števce in lastne množice vzorcev.

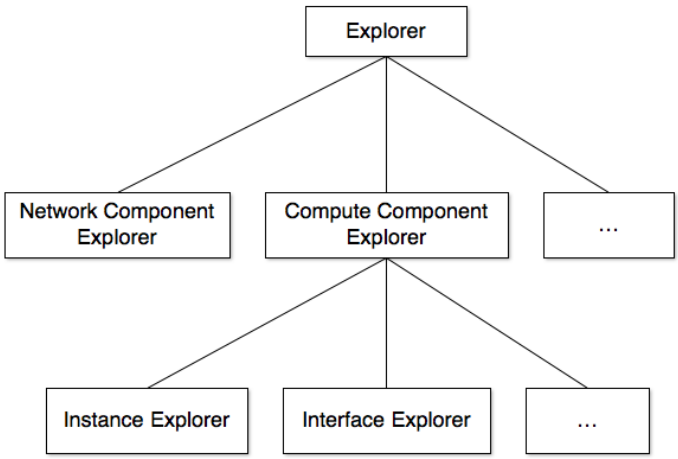
Identificiranje podatkov

Podatki o aktivnosti se med seboj precej razlikujejo. Vsaka komponenta ima lasten nabor virov, ki imajo ponovno lasten nabor števcev. Ti števci pa ustvarjajo svoje tipe vzorcev. Množica podatkov, ki jih lahko pridobimo, je tako precej velika, zaradi česar smo ustvarili postopek identificiranja in klasificiranja podatkov.

Urejanje podatkov smo osnovali na imenih števcev. Tej odločitvi botrujeta dejstvi, da so imena števcev unikatna[26] in da je podatek o imenu števca mogoče pridobiti iz vseh tipov podatkov, ki jih uporabljamo. Vsak vir namreč vsebuje podatke o števcih, ki so zanj na voljo. Podatek o svojem imenu vsebuje tudi vsak števec, vsi vzorci, ki jih ta števec izmeri, in vse statistike, ki jih Ceilometer izračuna na teh vzorcih. Z imenom števca lahko torej za vsak podatek ugotovimo, v katero hierarhijo podatkov o aktivnosti vira spada.

CeiloReader podatke ureja s pomočjo razreda *Explorer*. Ta razred je le začetek verige, dedujeta ga še dva nivoja razredov. Na drugem nivoju dedovanja so razredi, ki predstavljajo posamezne komponente. Na tretjem nivoju pa so končni razredi, ki predstavljajo dejanske tipe virov v komponentah. Posamezen objekt, ustvarjen iz razreda na tretjem nivoju, predstavlja konkreten vir. Na sliki 5.7 je prikazano drevo dedovanja iz razreda

Explorer. Če je klasifikacija podatka uspešna, se ustvari objekt razreda na tretjem nivoju. Glede na njegov tip in tip njegovih nadrazredov je natančno definirana njegova pripadnost. Če klasifikacija ne uspe, se ustvari objekt osnovnega razreda *Explorer*. Uporabnik bo podatke takšnega razreda še vedno videl, vendar se bo za njihovo interpretacijo in vizualizacijo vedno uporabila privzeta logika.



Slika 5.7: Dedovanje razredov *Explorer*.

Ko v aplikaciji v nekem pogledu prvič naletimo na nov nabor podatkov, s posebnimi metodami ustvarimo seznam *Explorerjev*. Objekte lahko ustvarimo iz poljubnega nabora podatkov, saj za uvrstitev podatka potrebujemo le ime števca. Za izbiro ustreznega razreda glede na ime štrevca uporabljamo vnaprej definiran seznam, v katerem so zapisane pripadnosti števcev ustreznim razredom. Del tega seznama je prikazan v izvorni kodi 5. Če podatka za določen števec v tem seznamu ni, njegova klasifikacija ne bo uspešna in ustvarjen bo osnovni razred. Podatke o imenih obstoječih števcev smo pridobili iz dokumentacije o meritvah Ceilometra.[22] Ob dodajanju novih števcev in razredov za njihovo interpretacijo je potrebno ta seznam posodabljati.

```
# List of explorer-meters matches.
EXPLORERS_METERS = [
    {
        'class': explorers.InstanceExplorer,
        'component_name': 'Compute',
        'meters': [
            'instance',
            'cpu',
            'cpu_util',
            # ...
            ]
    },
    # ...
]
```

Izvorna koda 5: Del seznama povezav med imeni števcev in razredi *Explorerjev*.

Uporaba razreda *Explorer*

Razred *Explorer* se uporablja v večini pogledov aplikacije CeiloReader. S pomočjo uvrščanja podatkov v ustrezne razrede se izvaja grupiranje na pregledih števcev in virov (prva nivoja zavihkov *Meters* in *Resources*). V obeh primerih se ustvarijo objekti *Explorer*, ki jih je mogoče združiti v ustrezne skupine.

Na drugem in tretjem nivoju obeh zavihkov pa se uporablja en ali več objektov *Explorer*. Vsak predstavlja svoj vir, za katerega z različnimi klici metod knjižnice Ceilometer Client pridobivamo podatke. Iz skupin podatkov vsak *Explorer* prebere podatke za svoj vir in jih zabeleži. Izvorna koda 6 prikazuje začetek identifikacije podatkov o statistikah nekega števca (drugi nivo zavihka *Meters*). Iz podatkov o statistikah najprej ustvarimo ustrezne objekte. Če je uporabnik izbral grupiranje po virih, potrebujemo dodatne podatke, saj so v podatkih o statistikah zabeleženi le identifikatorji ustreznih virov. Metoda *match_explorers_with_resources* zapiše nove podatke v ustrezne objekte *Explorer*. Podoben proces izvedemo za vse dodatne podatke, ki jih

pridobimo (podatki o uporabnikih, projektih...).

```
# Create explorers.
explorers = explorers_for_statistics(statistics_list=grouped_statistics,
                                     meter_name=meter_name)
# Fetch resource data if grouping by resource.
if group_by == 'resource':
    resources = [ceilometer.resources.get(resource_id=explorer.resource_id)
                 for explorer in explorers]
    # Match explorers with data.
    explorers = match_explorers_with_resources(explorers=explorers, resources=resources)
```

Izvorna koda 6: Uporaba razredov *Explorer* za identifikacijo virov v statistikah.

5.4.4 Podatki o uporabnikih in projektih

Podatke o uporabnikih in projektih pridobivamo od komponente Keystone. Ker so za upravljanje s temi podatki zahtevane administratorske pravice[21], so tudi v naši aplikaciji takšni podatki vidni samo administratorjem.

V izvorni kodi 7 je predstavljen proces pridobivanja podatkov o uporabnikih in projektih. Če je uporabnik aplikacije administrator, najprej pridobimo naslov Keystone dostopne točke. Objekt *User*, pridobljen med procesom overitve, namreč vsebuje seznam različnih dostopnih točk, odvisnih od njegovega položaja v hierarhiji uporabnikov. Ena izmed dostopnih točk administratorjev je tudi *admin_url*, na katerem je mogoče pridobivati podatke, namenjene administratorjem. V to skupino spadajo tudi tisti podatki, ki nas zanimajo v tem primeru. Če uporabnik ni administrator, uporabimo podatke, ki so nam na voljo. To so podatki o samem uporabniku in o projektih, katerih član je ta uporabnik. Ko podatke pridobimo, jih zapišemo v ustrezne objekte *Explorer*.

```
# Fetch projects and users if superuser. If not, use authorized tenants and active user.
if request.user.is_superuser:
    admin_url = find_identity_admin_url(request.user.service_catalog)
    keystone = keystone_client.Client(token=token_id, endpoint=admin_url)
    projects = keystone.tenants.list()
    if group_by == 'user':
        users = keystone.users.list()
        explorers = match_explorers_with_users(explorers=explorers, users=users)
else:
    projects = request.user.authorized_tenants
    if group_by == 'user':
        users = [request.user]
        explorers = match_explorers_with_users(explorers=explorers, users=users)
# ...
explorers = match_explorers_with_projects(explorers=explorers, projects=projects)
```

Izvorna koda 7: Pridobivanje podatkov o uporabnikih in projektih.

5.4.5 Specifična vizualizacija pregledov virov

Ker se lastnosti virov med seboj razlikujejo, je v aplikaciji mogoče definirati lastne poglede za posamezne tipe virov na drugem nivoju zavihka *Resources*. Ponovno uporabimo razred *Explorer*. Vsakemu razredu na tretjem nivoju je namreč mogoče definirati lastno logiko za prikaz pogleda. Vsebuje lahko dodatne klice za pridobivanje podatkov in specifične načine za njihovo urejanje ter uporablja lastno HTML predlogo. Podatki virov, uvrščenih v ta razred, bodo tako vizualizirani na njim primernejši način. Ko na začetku izvajanja privzete vizualizacije ustvarimo ustrezen *Explorer*, najprej preverimo, če vsebuje lastno logiko prikaza. Preverjanje obstoja in uporaba ločene vizualizacije sta prikazani v izvorni kodi 8. V nasprotnem primeru (če *custom_view* ne obstaja) se nadaljuje izvajanje privzete vizualizacije.

```
1  # Use custom view, if it exists.
2  if explorer is not None:
3      custom_view = explorer.custom_view(request, ceilometer)
4      if custom_view is not None:
5          return custom_view
```

Izvorna koda 8: Prikaz specifičnega pogleda razreda *Explorer*, če ta obstaja.

5.5 Težave med razvojem

OpenStack je aktiven projekt, ki se hitro razvija. Že med izdelavo diplomskega dela je prišlo do številnih sprememb v delovanju, izšla je celo nova izdaja. Številne funkcionalnosti je bilo mogoče zasnovati drugače, kot smo sprva načrtovali, kar je zahtevalo ponovno implementacijo nekaterih delov aplikacije.

Težave smo imeli tudi z občasno visokimi nalagalnimi časi. Za iskanje kritičnih elementov smo na drugem nivoju zavihka *Meters* implementirali merjenje časa izvajanja posameznih delov. Rezultati so pokazali, da največje zakasnitve povzročajo klici za pridobivanje podatkov, še posebej podatkov o virih. Da smo nalagalne čase zmanjšali, smo prilagodili funkcionalnosti, da so zahtevale čim manj klicev za identifikacijo podatkov. Počasno delovanje Ceilometra pa je opazila tudi skupnost, saj je bilo predlaganih (in izvedenih) precej optimizacij v delovanju komponente. Šlo je za različne pohitritve, ena izmed njih je na primer izboljšala hitrost SQL povpraševanj.[14]

Ker k projektu prispeva velika množica razvijalcev, smo občasno naleteli tudi na manjkajočo ali slabo urejeno dokumentacijo. Za razumevanje delovanja nekaterih elementov smo morali tako izvajati ločeno testiranje. Kljub temu nam to ni vzelo preveč časa, saj je skupnost trenutno na visokem nivoju in je mogoča tudi neposredna komunikacija z glavnimi razvijalci, ki so pripravljeni odgovoriti na različna vprašanja glede delovanja platforme.

5.6 Namestitev aplikacije

Aplikacijo CeiloReader je mogoče namestiti na lasten strežnik in jo uporabljati za spremljanje aktivnosti poljubne OpenStack namestitve.

5.6.1 Priprava okolja

Za namestitev na strežnik moramo najprej zagotoviti ustrezno okolje. Za delovanje aplikacije mora biti nameščen programski jezik Python verzije 2.7.5 ali novejši. Prav tako je potrebno namestiti ogrodje Django in spletni strežnik Apache z modulom *mod_wsgi*. Ta modul strežniku omogoča poganjanje Python kode in sodeluje pri delu s HTTP zahtevami.[8] Ker aplikacija uporablja tudi zunanje knjižnice, je potrebno namestiti še njih. Nekatere knjižnice so napisane v programskem jeziku Python in se namestijo neposredno na strežnik. Te knjižnice so:

django_openstack_auth 1.1.3 - Overjanje uporabnikov.

python-ceilometerclient 1.0.6 - Komunikacija s Ceilometrom.

django-bootstrap-breadcrumbs 0.6.2 - Navigacija po aplikaciji s pomočjo »drobtin«.

Za vizualno urejanje pa uporabljamo še nekaj drugih knjižnic. To so skupki CSS in JavaScript datotek, ki jih moramo shraniti na lokacijo, definirano v nastavitvah Django projekta. Gre za sledeče knjižnice:

Bootstrap 3.0.2 - Splošno vizualno urejanje aplikacije.

Flot 0.8.1 - Izrisovanje grafov.

bootstrap-datepicker 1.2.0 - Izbiranje datumov s pomočjo preprostega vmesnika.

5.6.2 Povezava z oblakom OpenStack

Ko je aplikacija nameščena, lahko komunicira s poljubnim OpenStack oblakom. Potrebno je nastaviti le dostopni točki za komunikacijo s komponentama Ceilometer in Keystone. Nastaviti ju je mogoče v splošni datoteki nastavitev projekta, predstavljata pa ju polji *OPENSTACK_KEYSTONE_URL* in *CEILOMETER_ENDPOINT*. Za pridobivanje podatkov o uporabnikih in projektih pa mora aplikacija imeti tudi dostop do naslova *admin_url*.

Poglavje 6

Nadaljnje delo

CeiloReader je že samostojna aplikacija, sposobna prikazovanja podatkov o aktivnosti oblaka. Za dobro delovanje pri veliki množici uporabnikov, ki spremljajo podatke obsežnega sistema v oblaku, pa je seveda mogoč še dodaten razvoj. Smer razvoja je odvisna predvsem od tipa končnih uporabnikov in oblaka, ki ga z aplikacijo nadzorujemo.

6.1 Izboljšanje uporabniške izkušnje

Če bi z aplikacijo spremljali oblak z velikim številom virov in uporabnikov, so priporočljive nekatere izboljšave grafičnega vmesnika in procesov pridobivanja podatkov. Za boljše nalagalne čase bi bilo potrebno uvesti prenašanje podatkov po delih. Tabelo s seznamom statistik bi na primer razdelili na strani, podatke za posamezno stran pa bi pridobivali, ko bi bilo to potrebno.

Podobne izboljšave bi lahko uvedli pri izrisu grafov, saj se trenutno vse krivulje hkrati izrišejo na isti graf. Pri velikem številu podatkov bi postali grafi hitro nepregledni. Te težave bi ponovno lahko reševali z delitvijo podatkov na manjše dele, uporabnik pa bi lahko izbiral, katere podatke želi imeti vizualizirane na grafu.

6.2 Uporaba novih funkcionalnosti

OpenStack in Ceilometer se konstantno razvijata in nudita nove, izboljšane načine pridobivanja podatkov. Že med razvojem diplomskega dela je postala mogoča uporaba novih, tako imenovanih kompleksnih povpraševanj. Kompleksna povpraševanja omogočajo podrobnejše definiranje zahtev za filtriranje in dovoljujejo izbiro logičnega operatorja, ki jih povezuje. Običajne zahteve za filtriranje so namreč vedno povezane s konjunkcijo, kompleksne zahteve pa so lahko povezane tudi z disjunkcijo.[26] Z uporabo kompleksnih povpraševanj bi lahko zmanjšali število zahtev za zbiranje in prenos podatkov, ki so tudi časovno najbolj zahtevne.

Izboljšani so tudi nekateri klici za pridobivanje podatkov. Ob pridobivanju podatkov o virih je tako mogoče določiti, ali želimo prejeti tudi podatke o števcih, povezanih z viri.[26] Če teh podatkov ne potrebujemo, je prenos, ki jih ne vsebuje, precej hitrejši.

Poglavje 7

Zaključek

V diplomskem delu smo definirali glavne pojme in tehnologije računalništva v oblaku. Ugotovili smo pomen beleženja in vizualizacije podatkov o aktivnosti v oblaku ter raziskali različne namene, ki jim ti podatki lahko služijo. Razvili smo spletno aplikacijo, ki ogromno množico podatkov, zabeleženo v komponenti Ceilometer, identificira in razvrsti. Iz velike množice podatkov z uporabo različnih načinov filtriranja prikaže zakonitosti, ki bi jih sicer zelo težko ugotovili. Uporabnik se lahko v aplikacijo vpiše z istim uporabniškim računom, kot ga uporablja za dostop do nadzorne plošče. Vidna mu bo tudi ustrezna množica podatkov glede na njegovo pozicijo v hierarhiji uporabnikov. Aplikacijo je seveda mogoče še nadgraditi in predvsem prilagoditi potrebam posameznih oblakov. Omogoča enostavno razširjanje z dodajanjem novih načinov vizualizacije za meritve, ki so v posameznem oblaku najpomembnejše. Znotraj okvirov trenutne verzije API-ja je sposobna razumevanja in vizualiziranja vseh tipov virov in njihovih meritev, tudi tistih, ki še ne obstajajo in jih bo Ceilometer ponujal šele v prihodnosti.

Menimo, da nam je uspelo predstaviti koncept aplikacije, ki uporabnikom nudi jasen pregled nad njihovo aktivnostjo v oblaku in vplivom te aktivnosti na sam oblak. S tem smo ustvarili rešitev, ki skupaj s Ceilometrom izpolnjuje zadnjo izmed petih lastnosti, ki definirajo računalništvo v oblaku - nadzor nad aktivnostjo.

Literatura

[1] "Aggregating statistics across instances," Amazon Web Services, Inc, 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/GetSingleMetricAllDimensions.html

[2] "Amazon CloudWatch," Amazon Web Services, Inc, 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: http://aws.amazon.com/cloudwatch/

[3] "Amazon CloudWatch pricing," Amazon Web Services, Inc, 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: http://aws.amazon.com/cloudwatch/pricing/

[4] "iCloud," Apple Inc., 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: https://www.icloud.com/

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, zv. 53, št. 4, str. 50–58, 2010.

[6] "The most popular front-end framework for developing responsive, mobile first projects on the web." 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: http://getbootstrap.com/

[7] J. Bort, "IBMhas brilliantly snatched away $2 billion cloud company from EMC," 2013, pridobljeno 23. 2. 2014. Na voljo na spletu: http://www.businessinsider.com/ibm-snatched-softlayer-from-emc-2013-6

[8] "How to install Django," Django Software Foundation, 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: https://docs.djangoproject.com/en/dev/topics/install/

[9] "Meet Django," Django Software Foundation, 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: https://www.djangoproject.com/

[10] "Dropbox," Dropbox, 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: https://www.dropbox.com/

[11] "Evernote," Evernote Corporation, 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: http://evernote.com/

[12] "Bossie awards 2013: The best open source data center and cloud software," InfoWorld, 2013, pridobljeno 23. 2. 2014. Na voljo na spletu: http://www.infoworld.com/slideshow/119863/bossie-awards-2013-the-best-open-source-data-center-and-cloud-software-226978

[13] R. Kiyanchuk, "OpenStack metering using Ceilometer," 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: http://www.mirantis.com/blog/openstack-metering-using-ceilometer/

[14] "The time of resource-list is too long," 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: https://bugs.launchpad.net/ceilometer/+bug/1264434

[15] "Will cloud computing overtake "on-premise" computing?" Market Info Group, 2013, pridobljeno 11. 5. 2014. Na voljo na spletu: http://www.marketinfogroup.com/will-cloud-computing-overtake-on-premise-computing/

[16] P. Mell in T. Grance, "The NIST definition of cloud computing," *National Institute of Standards and Technology*, zv. 53, št. 6, str. 50, 2009.

[17] "How to monitor a storage account," Microsoft, 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: http://azure.microsoft.com/en-us/documentation/articles/storage-monitor-storage-account/

[18] "How to monitor cloud services," Microsoft, 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: http://azure.microsoft.com/en-us/documentation/articles/cloud-services-how-to-monitor/

[19] "How to monitor web sites," Microsoft, 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: http://azure.microsoft.com/en-us/documentation/articles/web-sites-monitor/

[20] "Ceilometer developer documentation," OpenStack Foundation, 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: http://docs.openstack.org/developer/ceilometer/

[21] "Configuring Keystone," OpenStack Foundation, 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: http://docs.openstack.org/developer/keystone/configuration.html

[22] "Measurements," OpenStack Foundation, 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: http://docs.openstack.org/developer/ceilometer/measurements.html

[23] "OpenStack architecture," OpenStack Foundation, 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: http://docs.openstack.org/training-guides/content/module001-ch004-openstack-architecture.html

[24] "OpenStack projects, history, and releases overview," OpenStack Foundation, 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: http://docs.openstack.org/training-guides/content/module001-ch003-core-projects.html

[25] "System architecture," OpenStack Foundation, 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: http://docs.openstack.org/developer/ceilometer/architecture.html

[26] "V2 web API," OpenStack Foundation, 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: http://docs.openstack.org/developer/ceilometer/webapi/v2.html

[27] "Talligent," Talligent, 2014, pridobljeno 23. 2. 2014. Na voljo na spletu: http://talligent.com/

[28] J. Vičič in A. Brodnik, "Hybrid cloud monitoring," *Assisted ServiceS (CLASS)*, str. 47, 2012.